Cairo University
Faculty of Engineering
Computer Eng. Dept.
CMPS103

Data Structures and Algorithms
LAB#3
Stacks & Queues

Objectives

After this lab, the student should be able to:

- Use class templates to implement stack ADT (array-based and linked list-based).
- Pass and return Stack (template) to/from functions
- Use class templates to implement queue ADT (array-based and linked list-based).
- Pass and return Queue (template) to/from functions
- Implement the copy constructor for the ADT implanted as linked list.
- Write code to use stacks/queues in some common applications.

Part I – Stack ADT

A stack is a LIFO (**L**ast-**I**n-**F**irst-**O**ut) data structure that should support the following operations: push, pop, peek, and isEmpty

The Stack ADT interface is as follows:

```
//  This is an updated version of code originally
//  created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

template<typename T>
class StackADT
{
public:
   /** checks whether this stack is empty.*/
    virtual bool isEmpty() const = 0;

   /** Adds a new entry to the top of this stack.*/
   virtual bool push(const T& newEntry) = 0;

   /** Copies the top item of the stack to the passed parameter and removes it
from stack*/
   virtual bool pop(T& TopEntry) = 0;

   /** Copies the top item of the stack to the passed parameter without
removing it from stack*/
   virtual bool peek(T& TopEntry) const = 0 ;

   /** Destroys this stack and frees its assigned memory. */
   virtual ~StackADT() { }
}; // end StackADT
```

Stack Implementation:

StackADT can be implemented using array or linked list. Code example#1 gives an ***array-based implementation*** of the StackADT

💻 Code Examples #1

- Open "Stacks_Queues.sln" and run project "***1-Stacks***" and see how stack is implemented as class ***template*** using arrays:
  - ✓ Stack operations are implemented as class member functions
  - ✓ A check point is provided inside the code. Look for it and follow its instructions

Stack as a function parameter

Stacks can be passed/returned to/from a function. They can be passed by value or by references like any other data types.

💻 Code Examples #2

Run project "***2-Passing_Stack***" and see how stack can be passed and returned to/from a function.

a. Stack can be passed/returned by value, ref, const ref, etc.
b. Comment the copy constructor and re-run the example.
c. Can we use pass ArrayStack by const ref instead of pass by value?
   Modify function PrintStack to be

```
void PrintStack(const ArrayStack<T> &S).
```

   What is the problem?

Part II – Queue ADT

A queue is a FIFO (**F**irst-**I**n-**F**irst-**O**ut) data structure that should support the following operations: enqueue, dequeue, peek, and isEmpty

The queue ADT interface is as follows:

```
//  This is an updated version of code originally
//  created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

template<class T>
class QueueADT
{
public:
   /** Sees whether this queue is empty.*/
   virtual bool isEmpty() const = 0;

   /** Adds a new entry to the back of this queue.*/
   virtual bool enqueue(const T& newEntry) = 0;

   /** Copies the front item of the queue to the passed parameter and removes
it from queue*/
   virtual bool dequeue(T& FrontEntry) = 0;

   /** Copies the front item of the queue to the passed parameter without
removing it from queue
   virtual bool peek(T& FrontEntry) const = 0;
   /** Destroys this queue and frees its memory. */
   virtual ~QueueADT() { }
}; // end QueueADT
```

Queue Implementation:

QueueADT can be implemented using array or linked list. Code example#3 gives ***a linked list-based implementation*** of the QueueADT

Code Examples #3

Run project "***3-Queues***" and see how Queue is implemented as class template using linked lists.

- ✓ Queue is a class template and each Node in the queue is implemented by a class template as well.
- ✓ Two pointers are needed: a front pointer (where dequeue operation takes place) and a back pointer (where enqueue operation takes place).

Important Note:

As LinkedQueue is implemented using dynamic allocation:

- It should have a ***destructor*** to free the dynamically allocated memory
- It should have a ***copy constructor*** and should overload the ***assignment operator***. In the code example, the copy constructor is implemented whereas the assignment operator is left as an exercise for you to implement.

Queue as a function parameter and the copy constructor

Similar to stacks, queues can be passed/returned to/from a function. They can be passed by value or by references like any other data types.

Code Examples #4

Run project "***4-Passing_Queue***" and see how Queue can be passed to a function.

a. Why is LinkedQueue passed by reference to one function and by value to the other?
b. Check how function QueueSum takes a template specialization of LinkedQueue class (i.e. ***LiknedQueue<int>***) not the generic template (***LinkedQueue<T>***).

Practice Exercises

Note:

In the following exercises you are allowed to use only queues or stacks as auxiliary memory if needed. You are NOT allowed to modify queues and stack classes. You should use them as black boxes through their supported operations

Exercise 1

Extend the code of project "***1-Stacks***" and declare a stack of names and test its member functions on the new data type.

Exercise 2

Create a simple class Car with members: Car number and engine number.
Add setters and getters and write a program that

1- Uses ArrayStack class template to instantiate an object of ArrayStack that would store **pointers** to Car.
2- Create 3 Cars and fill with arbitrary values and push them into the stack
3- Pop cars form the stack and print their info.

Exercise 3

The given code example shows how to implement the stack using arrays
You are required to implement the stack using linked list instead of arrays.
Note: use class template ***Node*** given in Queue example.

Exercise 4

In code examples, Project " ***2-Passing_Stack***"
Update function Reverse to be ***void ReverseMe(…stack to be reversed…)*** so that the function reverses the passed stack itself and does not return anything.

Exercise 5

Write a function “collapse” that takes a stack of integers as a parameter and then collapses it by replacing each successive pair of integers by their sum.

For example:

- suppose a stack stores these values: bottom [**7, 2**, 8, 9, **4, 13**, 7, 1, **9, 10**] top

the stack should become: bottom [**9**, 17, **17**, 8, **19**] top

- and if stack is bottom [1, 2, 3, **4, 5**] top ➔ it becomes bottom[1, 5, **9**] top

Exercise 6

The given code example shows how to implement the Queue using linked lists
You are required to implement the Queue the using arrays.
What is the problem with array implementation?
What is circular queue? Implement it using array-based implementation

Exercise 7

Given a queue of sorted integers, write a C++ function that deletes duplicates from the queue **without changing the order of the remaining elements.** The original queue should be modified to contain only non-duplicates

Exercise 8

Write a function **SameOrder** that takes one stack and one queue and checks the contents of both of them is in the same order. Same order means that both stacks and queues have the same number of items and 1st item that **entered** the stack is the same as the 1st that **entered** the queue, and the 2nd is the same as the 2nd and so on.

Exercise 9

A ***palindrome*** is a string that can read backward and forward with the same result.
E.g., the following is a palindrome: **MADAM**
Further, a more general palindrome is one that ignores spacing, punctuation, and capitalization, such as the following is also a palindrome: **Go dog** and **Madam, I'm Adam**
Write a function that takes ***a string stored in a queue of characters*** and checks whether it is a general palindrome or not.
You are allowed to use Stacks only in your functions (char arrays and strings are not allowed)

Exercise 10

One of the applications of a stack is to **backtrack**—that is, to retrace its steps.
Write a program that:

1. Reads values from the user and stores them in a **queue** until zero is entered.
2. Scans the queue and does the following
    a. Each time a negative number is entered, the program must backtrack and print the five numbers that come before the negative number (starting from the one previous to the negative number) and then discard the negative number.
    b. If there are fewer than five items in the stack, print an error message and stop the program.
    c. The program repeats until the queue is empty or an error occurs

Test it with the following data:
Input Queue: **80** 1 2 3 4 5 **-1** **20** 30 40 50 **6 7 8 9 10** **-2** 11 12 -3 **33** **22** **-5**

Output: 5 4 3 2 1 **10 9 8 7 6** 12 11 50 40 30 **22** **33** **20** **80** Error: less than 5 items left